This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 397 924 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 89201248.5

(51) Int. Cl.5: G05B 19/417

(22) Date of filing: 17.05.89

(43) Date of publication of application: 22.11.90 Bulletin 90/47

(84) Designated Contracting States: DE FR GB IT SE

(71) Applicant: N.V. Philips' Gloeilampenfabrieken
Groenewoudseweg 1
NL-5621 BA Eindhoven(NL)

(72) Inventor: Biemans, Franciscus Petrus Maria
c/o INT. OCTROOIBUREAU B.V. Prof. Holstlaan 6
NL-5656 AA Eindhoven(NL)
Inventor: Sjoerdsma, Sjoerd
c/o INT. OCTROOIBUREAU B.V. Prof. Holstlaan 6
NL-5656 AA Eindhoven(NL)

(74) Representative: Strijland, Wilfred et al
INTERNATIONAAL OCTROOIBUREAU B.V.
Prof. Holstlaan 6
NL-5656 AA Eindhoven(NL)

(54) Work station controller module.

(57) A work station controller module for use in an industrial manufacturing system is described. The manufacturing system is hierarchically organized. A superior level is covered by a host machine, a lower level by automation modules that fulfill the manufacturing and transport primitives. The controller module receives higher level commands for decomposition into lower level commands. It also receives lower level status signals for aggregation to higher level status signals. It also exchanges product exchange requests and product exchange acknowledgements with a product transfer system. It further receives operation names and process data from a product development system and status aggregation rules and command decomposition rules from a product prepation system. These rules govern the decomposition/aggregation cited earlier.

EP 0 397 924 A1

Xerox Copy Centre

Work station controller module.

BACKGROUND TO THE INVENTION

The invention relates to a controller module for use in a work station, in particular, an industrial work station which is an element of a manufacturing control system. Manufacturing, in its broadest sense, encompasses various primitive, and undivided, operations on a product, such as test, storage or transformation. A work station thus comprises a controller module and one or more execution modules. An execution module may influence or observe the product in its environment; the product may thus be changed in this environment. A work station controller module as recited supra has been described by Scott and Strauss, "Work station control in a computer integrated manufacturing system", National Bureau of Standards, Washington, D.C., May 1984. Like the reference, the controller module of the present invention is adapted for use in transforming discrete products of all type. Transformation may be realized in manufacturing operations of all kinds, such as changing the product shape, assembling, modifying physical properties, sorting, testing, moving, packaging, combining and many others.

SUMMARY TO THE INVENTION

The inventors have realized that in such product transformation environments it is often possible to identify general tasks which are applicable to many different types of product transformation, so that the organization of the control could be independent of either product type or transformation type. Examples of such general tasks are:

- management of product availability (before, during and after transformation) which is tantamount to stock control;
- coordination among various transformation facilities, such as successive or parallel batch processing elements, merging and splitting of product streams;
- updating of various material properties (mechanical, dimensional, chemical, physical, electrical, etcetera);
- execution of the product transformations proper. In particular in this latter case, the controller module is rendered general-purpose by the extensive application of parameter/operation parametrization. Such parametrization may also be usefull to the three earlier aspects, but the embodiment hereinafter specifically relates to the product transformation parametrization.

In consequence, among other things it is an object of the invention to provide a work station controller that be applicable in general purpose situations, allows for a hierarchical transformation organization, may be used as a module in a multi-module system, is easy to reconfigure and to install, for a variety of products/transformations/ production scale. The object according to a first aspect thereof, is realized by the invention in that it provides a work station controller module for controlling a work station and comprising processor means interconnected to storage means for on the basis of predetermined operation-linked process data and under control of higher level commands and lower level status signals producing lower level commands and higher level status signals, said work station controller module further comprising:

- first bidirectional communication means for receiving said higher level commands from and transmitting said higher level status signals to a host device;
- second bidirectional communication means for receiving said lower level status signals from and transmitting said lower level commands to one or more product processing automation modules;
- third bidirectional communication means for transmitting product exchange requests and receiving product exchange acknowledgements with respect to any access point the work station has in common with a product transfer system or with at least one other work station;

first receiving means for receiving operation names and process data from a product development system; second receiving means for receiving status aggregation rules for aggregating said lower level status signals to said higher level status signals and command decomposition rules for decomposing said higher level commands to said lower level commands from a product preparation system.

Various further aspects of the invention are recited in dependent Claims.

Among other elements, the following are notable advantages of applying the invention:

- there exists a separation between the generation of the commands on the one side and the strategy of their execution, on the other; for one, this allows the host device to abstract from, or even ignore execution problems on a lower level;
- a work station controller module may interpret the strategy of execution that is presented in a recipe; for

each product there is a recipe, its execution depending on both the commands from the host and the actual status of the work station; this methodoly enhances flexibility;

- the execution proper of a command may be rendered dependent of both the applicable recipe, and the actual control by the higher level, such as represented by suspend and cancel commands, and their respective inverses;
- the work station controller module is standardizable through its general purpose applicability; it is suitable for integration because it has only few interfaces, it has excellent control features through operation, suspend, continue, cancel commands in addition to provision of status information and command accept acknowledgement, it has good reproducibility through commands for serially repetitive execution, and high performance through parallel execution of commands, safety through alarm handling capability, simplicity through exclusion of superfluous functionality, easy application-dependent applicability through input of recipes, of addresses of the transport system, of automation module entry points, and of an initial world model;
- the work station controller module can control multiple operational domains, provided only, that the processing facilities and/or storage capability were sufficiently therefor. Such multiple domains could relate to serially disposed automation modules in a single production stream (possibly separated by a transport facility or a temporary buffer), to correspondingly disposed automation modules in parallel production streams, or even to completely unrelation manufacturing/testing/etcetera operations. The internal organization could then occur on the basis of time-multiplex, job-multiplex, interrupt-driven or requests according to respective priority levels with associated arbitrage.

BRIEF DESCRIPTION OF THE FIGURES

The invention will be explained in detail with respect to the accompanying Figures, wherein

Figure 1 shows a conceptual environment of the work station controller module;

Figure 2 is a detailed block diagram of the work station and its close environment;

Figure 3 shows the functional set-up of the command decomposition versus the status aggregation.

DESCRIPTION OF AN OPERATION ENVIRONMENT

Figure 1 shows a conceptual environment of the work station controller module. Rectangular blocks represent functional entities. Blocks with rounded corners indicate the category of questions that the associated hardware and software level solves. Now, block 60 is the overall control system, herein the policy (62) is determined, such as product mix, quality levels, intermediate storage levels. One or more host subsystems 64, 66 interact with the overall control system and possibly, with each other. In general, the host level decides (68) where and when the various activities, such as product transformations are to be executed. Block 70 symbolizes the work station controller module. On a hardware level, this controller module is realized as a general purpose computer with appropriate processor means interconnected to storage means and suitable programming for executing its technical industrial task. A standard solution would be the Digital Equipment Corporations VAX machine series. Of late, IBM's computer PS/2 would be suitable. The interconnections between various subsystems in Figure 1 may be executed as point-to-point lines, as a single network such as VME, Ethernet, or with other means according to requirements. For brevity, no further explanation of the hardware is given. The work station controller 70 interacts with its own host 66, if required with other work station controller modules 71 and, among other things, comprises aggregation module 74. In general, the work station controller module solves the question how various changes should be effected, that is, to what quantitative and/or qualitative degree. On the lowest level, automation modules, to wit processing module (78) and transport module 80 directly interact with an identified product 84 or amount of product, and with their work station controller 70. The process identification is symbolized by block 82. As explained elsewhere, processing includes modifying the product, testing the product, repairing the product, detecting the product's position or orientation and other such primitive operations.

Figure 2 is a detailed block diagram of the work station and its close environment. The architecure is described in terms of 'processes': black boxes that accept and generate information.

If the workstation controller performs two activities in parallel, the architecture is described as a multiprocessing organization (single or plural procesors) each process executing one activity.

The workstation controller 22 is designed as a set of four parallel processes, the so-called H Module 27,

G Module 23, M Module 25, and A module 21. The functions of these modules are:

1. The H Module 27 accepts commands from a host 20, reports back to the host whether or not the commands are accepted or rejected, issues sub-commands to automation modules 24, 26, 28 and messages to the transport system 30, 32 or to other work station controllers to negotiate the exchange of products. To that end, the H Module, inter alia:

- accepts commands.
- when required, executes recipes which prescribe which actions have to be carried out by the automation modules in order to fulfil commands.
- manages recipes.

2. The M Module 25 maintains a model description of the subordinate automation modules 24, 26, 28 and that part of the environment that is within the domain of the work station. That model ('world model') contain such information as the configuration of automation modules, their capabilities and capacities, the products and tools inside the domain of the work station. Such information is used, for example, by the H Module to decide to which automation module a command can best be issued.

3. The G Module 23 collects and processes/aggregates status signals from the automation modules 24, 26, 28 in order to keep the world model of the M Module as consistent as possible with the real world. The G Module of a work station controller ensures, for example, that the world model contains up-to-date information on the products that are present in the workstation's domain (which products and where). The G Module also provides the host 20 with status signals. The status offered to the host is so called 'aggregate status': status about the workstation in total as opposed to status of individual automation modules. Suppose that one automation module has a defect so that the workstation can no longer execute a certain operation. The manager system 60 should only be informed that the workstation can no longer execute that operation; it does not have to know which automation module is the cause. The G Module also has to establish whether or not some status gives rise to an alarm. In case of alarm, the G module suspends the H module in that commans are accepted from the host, but lower level commands are no longer given to the automation modules, as long as the alarm prevails.

In addition to these three modules, there is a manager A, 21. The manager offers to H, M and G Modules 23, 25, 27 parameter values that are characteristic for a particular application and that may vary in time.

In Figure 2, the various 'streams' of communications are indicated by 'gates', letters. Processes that have one or more gates in common can communicate, as follows:

* = update, status request
m : values from manager to H Module, or values from manager to G Module, or values from manager to M Module.
a : messages from/to automation modules (38).
t : messages from/to transport system (42).
w : messages from/to host (36).
r : recipes from development system (31).
f : formulae from development system (29).
s : status to and from M Module

Finally, the Figure 2 shows the product 34 as being operated by the automation modules 24, 26, 28 (inclusive of test module 28) and transport system 30, 32.

The H module 27 is initialized in that it receives the following application-dependent data from host 20:

- Its potential activity. This is a set of Operation Descriptors, that are activated when the controller module itself is activated. Some are selfterminating, others would remain active permanently.
- The autonomous operations. This is a further set of Operation descriptors, that are activated when the controller module itself is activated. Some are self- terminating, others would remain permanently active.
- The Partners. This is a set of addresses of its automation modules 24, 26, 28 and the address of the work station controller. After receiving these information the H module is controlled for thereby upstarting the whole work station controller 22.

2. The H module behaves as three concurrent processes:

a. - ACCEPT COMmands process: Operation Commands, Suspend Commands, Continue Commands and Cancel Commands destined for the workstation are received from host 20. The Operation Commands are either accepted or rejected. The accepted ones are kept for execution. The Suspend, Continue and Cancel Commands are executed. An Operation Command is passed to EXECute COMmandS process for execution.

b. - EXECute COMmandS process: The Operation Commands are executed. EXECute COMmandS will process and consider recipe lines for evaluation and execution.

c. - RECIPE MANAGER process: The recipes are maintained, updated or deleted.

The latter three activities are discussed in more detail below.

ACCEPT COMmandS maintains the following data for the purposes of administration:

- A set of Operation Commands. The set of Operation Commands contains the Operation Commands that are executed by the work station. The Repeat Number of the Operation Command indicates the number of Operations that can still be suspended or cancelled.

Suppose, for example, that an Operation Command be executed 10 times. If the work station controller has started the execution of the first Operation out of the series of ten, nine Operations can still be cancelled or suspended. The Operation Command should then contain the repeat number '9'.

- A first set of Entry Codes of the operations that are suspended.
- A further set of Entry Codes Entry of the operations that are cancelled.

ACCEPT COMmandS:

1. either, receives an Operation Command destined for the work station. It then inspects the Potential Activity to decide whether or not the operation can be executed. If it cannot be executed, the process ACCEPT COMmandS will return a Rejection Indication to host 20. The Rejection Indication contains the Entry Code of the rejected Operation Command. If the operation is accepted, the process ACCEPT COMmandS concurrently:

- responds to the host with an Acceptance Indication which contains the Entry Code of the accepted Operation Command.
- issues the Operation Description and Entry Code contained in the Operation Command to the process EXECute COMmandS, for execution of the operation. Next, a new Operation Command is inserted in the set of Operation Commands.

2. or, receives a Suspend Command destined for the work station. It then inspects the set of Operation Commands to establish whether or not the Operation Command with the same Entry Code as the Suspend Command can still be suspended because of the number of operations to be executed one or more had not been started yet.

- If so, it returns a Suspend Indication, which contains an Integer that states how many times an operation will be suspended. This integer can be derived from the Operation Command with the correct Entry Code in the set of Operation Commands (repeat number). Next, the module inserts the Entry Code in the set of Entry Codes of suspended operations.
- If no suspend is possible, it returns a Suspend Indication with parameter zero.

3. or, receives a Continue Command destined for the work station. It will then remove the Entry Code contained in the Continue Command from the set of Entry Codes of Suspended Operations. If this Entry Code is not present there, this set remains unchanged.

4. or, receives a Cancel Command destined for the work station. It then inspects the set of Operation Commands to establish whether this command may be cancelled because of the number of operations to be executed (one or more) had not been started yet.

- If so, it returns a Cancel Indication. The Cancel Indication Contains an Integer that states how many times an operation will be cancelled. This repeat number is derived from the set of Operation Commands. It then inserts the Entry Code in the set of Entry Codes of cancelled operations and removes (if present) the Entry Code from the set of Operation Commands and removes the Operation Command concerned from the set of Operation Commands.
- If cancelling is impossible, it returns a Cancel Indication with the integer zero.

5. or, it receives from the process EXECute COMmandS, which is executing accepted Operation Commands, a request to report whether or not a certain Entry Code belongs to those Operation Commands that

- should still be executed or are completed (with respect to the repeat number that tells how often an execution should be repeated), or
- have been suspended, or
- have been cancelled.

Having reported this, ACCEPT COMmandS,

o if the operation has been cancelled and the repeat number is still greater than zero, removes the Entry Code from the set of cancelled Entry Codes.

o if the operation has not been cancelled, and has not been suspended, and the repeat counter is still greater than zero, decrements the repeat number.

o if the operation has not been cancelled, has not been suspended and the repeat counter is less than or equal to zero, issues a Progress Indication to the host with the progress status 'ready', and removes the Operation Command from the set of Operation Commands, and the Entry Code from the set of suspended and cancelled Entry Codes.

o if the operation has not been cancelled, the repeat number is still greater than zero, but the operation has been suspended, does not change the set of Operation Commands, nor the sets of suspended and cancelled Entry Codes.

6. or, ACCEPT COMmandS receives an indication that the execution of an Operation with a certain Entry Code has been blocked from the process EXECute COMmandS (meaning that the search for a Recipe Line that can be processed takes too much time).

- ACCEPT COMmandS searches for the Operation Command with this Entry Code in the set of Operation Commands, and then issues a Progress Indication to the host 20 that contains the Entry Code and the Progress Status 'blocked'.

After execution of any of steps 1 through 6, a next step can be executed thereof.

EXECute COMmandS

The process EXECute COMmandS :

1. Either, receives an Operation Description and an Entry Code from ACCEPT COMmandS,

2. or takes one of the Operation Descriptions of the set of autonomous operations; then receives a Recipe suitable for the execution of the Operation Command. Any Recipe is accepted from the process RECIPE MANAGER, if that the Operation Descriptions Recipe and of the Operation Command are equal. EXECute COMmandS then makes the actualised Recipe available for the Operation Command, then executes this Recipe as many times as required.

3. Concurrently, accepts an Operation Command and processes these as discussed in item 1 and 2. (Hence, the work station controller can execute an infinite number of commands in parallel).

The Execution Of A Recipe

Below, the execution of a recipe, for the fulfillment of a commanded operation is discussed first. Next, the execution of a recipe for an autonomous operation is discussed.

1. First, ACCEPT COMmandS is asked (by reference to the Entry Code) whether the operation is cancelled, suspended or should still be repeated.

- If cancelled, the execution of the recipe stops.
- If the operation should be no longer repeated, the execution of a recipe stops.
- If suspended, the behaviour detailed here is repeated.
- If not cancelled, nor completed nor suspended, the execution continues.

2. If the execution of a recipe is continued, a Recipe Line suitable for processing is selected.

1. All lines of the Recipe with the required Step Number are candidates for processing. When the execution of a Recipe is started, a line with Step Number 'init' is searched for. There is a set of lines with the correct step number. One line of them has to be executed. First, one of them, suitable for execution is selected for evaluation.

First, the M MODULE is asked whether or not there is an alarm. If so, the activities detailed under 2 (here) are repeated. Else, evaluation of Recipe Lines continues.

The M MODULE is dedicatd exclusively, so that only the process EXECute COMmandS has access to the world model for the processing of this recipe line. The set of conditions of this line and its Update Status Instructions and its Move Status Instructions are passed to the M MODULE. This process checks whether or not all conditions of the recipe line with respect to the contents of the world model are true. If these conditions are true, the M MODULE 25 will execute the (recipe line) Update Status and Move Status Instructions. Finally, the M MODULE will report whether or not the conditions were true (and consequently, whether or not the instructions were executed).

Next, the Send Message Instructions of the recipe line will be executed by the process EXECute COMmandS in an arbitrary order.

EXECute COMmandS will send each message to its destination: If the message is destined for an automation module, EXECute COMmandS also requests the M MODULE 25 to accept a Status element with the variable 'receiver of the message, the operation the message is referring to' and one of the

following values:
o if the message is an Operation Command, the value 'issued'.
o if the message is a Reset Command, the value 'reset'.

If the message is destined for the transport system, EXECute COMmandS also requests the M MODULE to accept an element with the variable 'accesspoint' and one of the following values:
o if the message is a 'Control Passed' message, the value 'noBody'.
o if the message is a 'Control Workstation Transport System Request' message, the value 'issued'.

If the message is destined for the host 20 (progress indication or status indication), this message is sent to the host without updating the world model. Next,
- if the next step number of the executed line is 'exit', no further recipe line is executed.
- if the next step number of the executed line is not 'exit', the activities detailed under 2.1 are repeated with the group of lines with this next step number.
- If the conditions were not fulfilled, the instructions had not been executed, and a subsequent line (possibly the same) with the same step number is considered for evaluation and execution.

2. If, after a certain time out period, no Recipe Line has been selected, EXECuteRECipe may decide to issue an indication to ACCEPT COMmandS to report that the operation with a certain Entry Code is blocked. The searching for a Recipe Line as described above continues.

The processing of a recipe for autonomous operations occurs in a similar way, except for:
- suspension, canceling and repetition of the execution of a recipe are not considered.
- there is no time out period.

The recipe manager has to:

1. Either, process a recipe update request.
- receive an update request (from the product preparation sysem) that contains a recipe. If this recipe replaces an existing recipe because it has the same recipe description, delete the existing one and insert the recipe contained in the update.

2. Or, process a recipe delete request.
- receive a delete request (from the product preparation system) that defines the recipe identification of the recipe that should be deleted.
- then, if a recipe with this identification is present, delete the recipe.

3. Or, delete one recipe on own initiative.
- delete one of the recipes managed by the recipe manager. This is to control the deletion of recipes due to limited memory space.

4. Or, offer the process EXECute COMmandS a recipe that is required by the process EXECute COMmandS.

When any activity (1, 2, 3 or 4) is finished, RECIPE MANAGER can execute a subsequent one.

The G MODULE:

1. first initialises. The following application-dependent dat are received from the WORKSTATION CONTROLLER MANAGER, 21
- a set of Status Request Information Tuples, which prescribe the status that should be monitored by the workstation controller.
- The Partners. This is a set of addresses of its automation modules, the address of the transport system and the address of the workstation controller itself.
- The locations. These are the Variables of status elements that describe the places at which products or tools may reside.
- The access points where products and tools may enter or leave the workstation controller area.

2. then, behaves as five concurrent processes:
- RECeiVe STATus and EXECute STATus: Status Indications from Automation Modules are received by RECeiVe STATus and presented to EXECute STATus. EXECute STATus updates the world model and determines which aggregated status is to be sent to the host and/or world model. The relationship between received status and other statuses kept in the world model and aggregated status is given in the formulae.
- In addition, RECeiVe STATus issues Status Requests to Automation Modules.
- FORMULA MANAGER:
The formulae are maintained, updated or deleted.
- PROVIDE STATus Status Requests from the host are responded to
- TRANSFER RESOURCE CONTRol:

Messages are exchanged with the transport system to control product transfer between work station and transport system or between two work stations.

RECeiVe STATus, EXECute STATus, PROVIDE STATus, FORMULA MANAGER and TRANSFER RESOUCE CONTRol are detailed in the following sections.

RECeiVe STATus

Using the set of status request information tuples and the addresses of the automation modules and the work station itself as application-dependent data, the process RECeiVe STATus :

1. Issues Status Requests. It inspects the set of Status Request Information Tuples, and uses the Variable and Address in one Tuple to generate a Status Request to an Automation Module.

2. Receives Status Indications. It receives a Status Indication destined for the work station, and presents these to EXECute STATus. In some cases, a special Status Element is generated with the Variable 'sender of the message, the operation that is referred to' and the value:

- if the message is an Acceptance Indication: 'busy'.
- if the message is a Rejection Indication: 'rejected'.
- if the message is a Progress Indication: the status of progress.

EXECute STATus

EXECute STATus receives Status from RECeiVe STATus and requests the M MODULE to update its world model according to this status. Next, the Formula defining the relations between most recently received status and existing status is selected. The formula contains several formula lines. Each formula line consists of one or more conditions and one or more instructions. Each of these lines is evaluated as to whether or not all conditions in the line are fulfilled. The evaluation of one formula line is as follows: EXECute STATus locks the world M MODULE to ensure exclusive access to the world model.

The set of conditions of the formula line, its Update Status Instructions and its Move Status Instructions are passed to the M MODULE. This process checks whether all conditions of the formula line with respect to the contents of the world model are true.

- If so, the M MODULE will execute the Update Status and Move Status Instructions. Finally, the M MODULE will report whether or not the conditions were true (and consequently, whether or not the instructions were executed).

Next, the Send Message Instructions of the formula line will be sent in an arbitrary order to the host by the process EXECute STATus. Next, EXECute STATus unlocks the M MODULE.

- if any condition is not true, the M MODULE is unlocked.

Alarm Handling

Alarms can be considered as special cases of aggregate status. Alarm messages if needed have to be incorporated in the set of Send Message Instructions of Formulas.

FORMULA MANAGER

The formula manager has to:

1. Either, process a formula update request.

- receive an update request (from the product preparation system) that contains a formula.
- then, if this formula replaces an existing formula because it has the same formula description, delete the existing one and insert the formula contained in the update.

2. Or, process a formula delete request.

- receive a delete request from the product preparation system that defines the formula identification of the formula that should be deleted.
- then, if a formula with this identification is present, delete the formula.

3. Or, offer the process EXECute STATus a formula that is required by the process EXECute STATus.

When activity 1, 2 or 3 is finished, FORMULA MANAGER can again execute either activity 1, 2 or 3.

PROVIDE STATus

Using the locations and the address of the host as application-dependent data, PROVIDE STATus:

1. receives a Resource Request destined for the workstation.

- Then, PROVIDE STATus requests the M MODULE to report the identifiers of resources that are located at the positions of the 'locations'.
- Then, PROVIDE STATus counts the number of identifiers that equal the identifier contained in the Resource Request, and sends this number to the host in a Resource Indication Message.

TRANSFER RESOURCE CONTRol

Using the access points, the locations and the addresses of work station controller module and transport system as application-dependent data, TRANSFER RESOURCE CONTRol simultaneously controls each of the access points:

1. either, it processes control transfer requests from the transport system:

- receives a control transfer request from the transport system directed to the work station controller module concerning the access point.
- then, locks the world model and asks the world model manager to insert status that states that the access point is claimed by the transport system, provided that in the world model the access point is claimed by nobody.
- then, receives from the world model manager, a reply that states whether or not the conditions for updating were true and unlocks the world model.
- If not, a control refused message concerning the access point is returned to the transport system.
- If so, a control granted message concerning the access point is returned to the access point.
- Or, TRANSFER RESOURCE CONTRol receives a control passed message concerning the access point.

TRANSFER RESOURCE CONTRol then updates the world model with status that states:

- the name of the product, as contained in the control passed message, that resides at the location of the entry point and associated product data.
- the fact that no element is claiming control of the access point.

2. or, receives a control granted message from the transport system concerning the access point. Next, TRANSFER RESOURCE CONTRol updates the world model with status that states that the control of the access point has been granted.

3. or, receives a control refused message from the transport system concerning the access point. Next, TRANSFER RESOURCE CONTRol updates the world model with status that states that the control of the access point has been refused, and continues to behave as detailed in 1, 2 or 3.

The M MODULE 25:

1. first initialises. It receives the application-dependent initial world model from the work station controller manager 21.

2. Then, it behaves as WORLD MODEL MANAGER. It manages the world model by processing requests for status and updating the model with Status received from both the H and G MODULE.

WORLD MODEL MANAGER

The WORLD MODEL MANAGER has to:

1. either, receive and process update requests.

It has been described in sections EXECute COMmand and EXECute a STATus that updates on the world model also depend on the current state of this world model. Updating the world model is enabled to at most one participating process at a time to guarantee the consistency of the world model. This is done as follows :

1. first receive a lock request

2. next, 1. Either receive and process an Update Status Request, that specifies a status element

that should be inserted in the world model. Then, insert this status element and remove the existing status element, if any, with the same status descriptor as that of the status element contained in the update request.

2. or receive and process a Move Status Request. Assign the Value of the Status element with the first Variable to the Status Element with the second Variable. The Status element with the first Variable receives the Value 'noResource'.

3. or receive and process conditions and update status commands and indicates whether or not these conditions were fulfilled

- receive a set of conditions and a set of status elements and return a 'true' if the conditions are true with respect to the world model a, "false" if not. If true, the world model is updated according to the new status elements.

4. or receives and processes conditions and update status or move status commands. If the conditions are true, the world model is updated according to the new status elements.

2. or, receive and process request for one or more Status elements: receive a status request that specifies a status Variable or a set of Variables, then supply the requested status elements.

FURTHER ORGANIZATIONAL DESCRIPTION

Figure 3 shows the functional set up of the command decomposition versus the status aggregation. Three successive hierarchical levels are shown. The upper one 100 may control the complete production line from raw material to finished product. The second level 102 controls the unit product transformation. The third level 104 modifies the respective physical entities. The work station controller module 102 is on the middle level. Each level has its own world model, which is restricted to the aspects relevant to the level in question. For example, the lower level knows physical and geometrical aspects of each separate product present. The next higher level could know the number of products present without knowing physical details. As the situation may require, the number of successive layers may be higher or lower. On their own respective level, the world models communicate with command decomposition elements 130, 132, 134 and status aggregation elements 110, 112, 114. On each level a composite command may be separated into more elementary commands. On each level a number of more elementary status signals may be aggregated to more composite status signals. For example, an aggregated alarm may be ORed from a plurality of elementary alarms. A plurality of elementary stock feed requests is added to a composite stock feed request by adding the requested quantities.

MESSAGES

Hereinafter, messages are discussed systematically between the host and the work station controller

name	communicators	entry code	further :
operation command	x	x	operation description, repeat counter
suspend/continue/cancel commands	x	x	-
acceptance/reject.ind.	x	x	-
suspend/cancel ind.	x	x	repeat counter
resource request	x		resource identif.
resource identifier	x		resource ident., integer
progress indication	x	x	status of progress
status request	x	x	status variable none
status indication	x		status

Herein, the -Communicators- define the sender and receiver of the Message.

1. The Operation Command commands an execution of an operation on resources, and may indicate one or more products and one or more operations.

- The Repeat Counter defines how many times the execution must be repeated.
- The Entry Code facilitates a reference to an Operation Command.

Some relationships with other messages:

- An Operation Command is accepted or rejected by the work station controller module. In the case of acceptance, the latter returns an Acceptance Indication with the Entry Code for reference to the accepted Operation Command. In the case of rejection, a Rejection Indication is returned that contains the Entry Code.
- Suspend Command, Continue Command and Cancel Command temporarily stop, restart and permanently stop the execution of the operation, respectively.

2. The Suspend Command is used to temporarily stop the execution of an operation, in the sense that during the suspension no resources (materials/tools) should be delivered by the workstation.

Parameters:

- The Entry Code refers to the Operation Command, whose execution is to be suspended.
- The work station controller reacts to a Suspend Command with a Suspend Indication.

3. The Continue Command is used to cancel certain Suspend Commands so that the execution of any suspended operation is continued.

Parameters:

- The Entry Code refers to the Operation Command, the execution of which is to be continued.
- The Continue Command deletes a Suspend Command that has the same Entry Code.

4. The Cancel Command is used to prevent further execution of certain Operation Commands. If the operation has not yet started, it will no longer be started. In case the Operation Command commanded a series of operations, that part of the series that has not yet started will no longer be executed.
- The Entry Code refers to the Operation Command, the execution of which is to be cancelled.

Some relationships with other messages:

- The Cancel Indication reports the cancellation.

5. The Acceptance Indication is used by the work station to indicate that it can execute a certain operation. The operation can be executed if it does not conflict with the (permanent) ability of the work station. The presence of tools or materials is not evaluated.
- The Entry Code refers to the Operation Command that is accepted.
- The work station controller may also respond to an Operation Command either with an Acceptance Indication or with a Rejection Indication.

6. The Rejection Indication is used by the work station to indicate that it cannot execute a certain operation because of a permanent inability of the work station. A temporary shortage of tools or materials is not signalled.
- The Entry Code refers to the Operation Command, the execution of which is rejected.

Some relationships with other messages:

- The Rejection Indication may be a response to an Operation Command as alternative to an Acceptance Indication.

7. The Suspend Indication is used by the work station to report its fulfilment of a Suspend Command.
- The Entry Code refers to the Operation Command, the execution of which has been suspended.
- The Repeat Counter indicates how many operations have been suspended, for possible later execution.

8. The Cancel Indication is used by the work station to report its fulfilment of a Cancel Command.
- The Entry Code refers to the Operation Command, the execution of which has been cancelled.
- The Repeat Counter indicates how many operations have been cancelled.

9. The Resource Request is used by the host to inquire about the number of certain resources (materials, tools) present within the domain of control of the work station.
- The Resource Identifier defines the resource.

10. The Resource Indication is used by the work station to inform the host about the number of

certain resources (materials, tools) present within the domain of control of the work station.
- The Resource Identifier defines a resource.
- The Integer indicates the number present.
- The Resource Indication can be either a response to a Resource Request, or a non-solicited message, generated by the work station.

11. The Progress Indication is used by the work station to inform the host about the progress of the execution of certain operation commands. The execution can be ready or temporarily blocked, for example because of a shortage of materials or tools.
- The Entry Code refers to an Operation Command, of which the execution progress is reported.
- The Status of Progress (ready/blocked) defines the progress.

12. The Status Request is used by the host to be informed of (part) of the status of the work station.
- The Variable refers to the required Status.
- The Status Indication reports the Status Variable and the actual Status Value.

13. The Status Indication is used by the work station to inform the host about a certain status. Status Indication can be provided on command of the host or spontaneously.
- The Status Indication contains a Status Element.

2.2 WORK STATION CONTROLLER - AUTOMATION MODULE

The following Messages are exchanged between work station controller module and automation module:

Name of the message	communicators	operation	further
OPERATION COMMAND	x	x	
RESET COMMAND	x	x	
ACCEPTANCE INDICATION	x	x	
REJECTION INDICATION	x	x	
PROGRESS INDICATION	x	x	status of progress
STATUS REQUEST	x	x	status variable
STATUS INDICATION	x	x	status element

1. The Operation Command is used to command the execution of an operation.
- The Operation Identifier defines the operation to be executed.

Some relationships with other messages:

- An Operation Command is accepted or rejected by the automation module controller, which returns an Acceptance Indication or a Rejection Indication, respectively.
- Reset Command can stop the execution of the operation by returing the automation module to a predefined state.

2. The Reset Command is used to disrupt the execution of an operation. The automation module returns to a pre-defined state.
- The Operation Identifier refers to the Operation, the execution of which is to be disrupted.

3. The Acceptance Indication is used by the automation module to indicate that it can execute a certain operation because of permanent ability.
- The Operation Identifier refers to the Operation Command, the execution of which is rejected.
- The Acceptance Indication is a response to an Operation Command.
- The automation module may alternatively respond to an Operation Command with a Rejection Indication.

4. The Rejection Indication is used by the automation module to indicate that it cannot execute a certain operation, because of a permanent inability.
- The Operation Identifier refers to the Operation Command, the execution of which is rejected.

5. The Progress Indication is used to report the progress of an operation. The progress can be:
- 'ready', when the operation is completed, or
- 'blocked', when the operation is temporarily suspended because of missing materials or tools.

6. The Status Request is used by the work station controller to be informed of (part) of the status of

the automation module.
- The Variable refers to the required Status.

7. The Status Indication is used by the automation module to inform the work station controller about a certain status. Status Indication can be provided either on the command of the work station controller or self-reliantly.
- The Status Indication contains a Status Element.

2.3 WORKSTATION CONTROLLER - TRANSPORT SYSTEM

The work station controller and transport system communicate in order to transfer the control of resources (materials, tools). The following control transfer messages are used.

Name of the message	communicators	access point	further
CONTROL TRANS WORKST REQUEST	x	x	
CONTROL WORKST TRANS REQUEST	x	x	
CONTROL GRANTED	x	x	
CONTROL REFUSED	x	x	
CONTROL PASSED	x	x	product, product Data

The Communicators define the sender and receiver of the Message. The Access Point defines a place where products or tools may enter or leave the work station controller's domain of control.

Short description of the messages:

1. The "Control Trans Workst Request" message is used by the transport system to ask the work station controller permission to pass control of a product or a tool from the transport system to the work station controller.

2. The Control Workst Trans Request message is used by the work station controller to ask the transport system permission to pass control of a product or a tool from the work station controller to the transport system.

3. The Control Granted message is used by a sender to grant permission to pass control of a product or tool from the receiver to the sender.

4. The Control Refused message is used by a sender to refuse permission to pass control of a product or tool from the receiver to the sender.

5. The Control Passed message is used by a sender to acknowledge that control of a product or a tool has been passed to the sender.

In addition to communicators and access points, the Control Passed Message has two parameters:

1. Product: This concerns all information that is available to the system that has the product under its control. An example of such information is the name of a product.

2. Product Data: This is information so closely associated with a product that the sender of the data requests the transport system to transfer the data simultaneously with the product.

APPLICATION-DEPENDENT DATA

The work station controller is tailored to its application by provision of pertinent data. In particular, 'recipes' prescribe how the work station controller should execute certain operations, selection of certain materials, a certain configuration of automation modules in its domain etc.

Application-dependent data as input are separated from the command and status interfaces with the host, automation modules and transport system. The reason is that the provision of the application-dependent data is not the responsibility of the host. The host determines what is to be done by the work station. It does not determine how it is to be done. The host, for example, commands the execution of a

particular operation; the recipe prescribes how this operation should be executed.

'Partners' are sets of addresses or names of the subsystems that should communicate. These are, the address of the work station itself, of the automation modules, of the transport system and of the host.

LOCATIONS are a set of Variables of Status elements that refer to places where the work station controller can locate such resources as material and tools. The work station controller maintains Status elements that consist of these Variables and of Values that describe the Identity of the product or tool at that location.

ACCESS POINTS are variables referring to the locations at which products can pass from the transport systems domain of control to the work station's domain of control or vice versa. These locations are controlled by both the transport system and the work station controller. The variables are used by the workstation controller itself, and can also be used in recipes to assign the control of a location.

A WORLD MODEL is contained in the work station controller and gives a model of the state and capabilities of its subordinates. Among others, this is needed to reflect the global state of the work station. Depending on this state, the work station controller may decide to execute different actions.

At start-up, an initial world model is fed into the work station controller. During operation, the work station controller updates this model.

The model is a set of Status elements. Each Status element consists of a Variable and a Value, for example, a location and a product name, saying that a Product is present at a Location.

SET OF STATUS REQUEST INFORMATION TUPLES

Because the work station controller should update its world model, it should know which information it should collect for this purpose. The updating may occur by issuing appropriate status requests to automation modules. Therefore, the work station controller needs to receive information that says which types of status it has to collect from which automation module. This is defined by a set of so-called Status Request Information Tuples. A Status Request Information Tuple contains a Variable and an Address of an automation module.

The work station controller will issue requests to the systems at these addresses to supply the Status element belonging to the Variable.

3.6 RECIPES

A recipe is a presciption of how a work station controller should execute a command. A Recipe consists of a Recipe Identification and a sequence of one or more Recipe Lines.

A Recipe Identification is an Operation Description which defines the Operation for which the Recipe can be used. A recipe line consists of:

|Step Number |Conditions |Instructions |Next Step Number|

1. A step Number identifies a step in the execution of a Recipe. A Recipe Line specified by this Step Number will be executed. There may be more than one Recipe Lines with the same Step Number. However, only one of the Recipe Lines can be selected for further execution. Which one will be executed depends on its Set of Conditions as will be explained below.

2. A set of Conditions is an assertion about a Status Variable and Value. For example 'temperature equals 100' consists of a Variable, an Operator and a Norm Value.

The Variable refers to a Status element in the world model of the work station controller. The work station controller compares the Condition with the Status element of the world model, and determines whether or not the Condition is satisfied, given the Value of the Status.

The Operator can be an equal (=) or not-equal (#) to the operator and can act upon all sorts of Values. It can also be a less than (<) or greater than (>) operator provided that the Values can be ordered.

A Recipe Line can only be executed if all Conditions of its set of Conditions are met. The Conditions may define more than one assertion about the same Variable, for example:

temperature < 100	"temperature less than 100"
temperature > 90	"temperature greater than 90"
location1 = productx	"at location1 is products"

3. The Set of Instructions are those that should be executed if the conditions of the recipe line are true:

- Send Message Instruction; the Recipe Instruction consists of a Message that should be sent to its destination (automation module, host or transport system). Such a message could be, for example, an Operation Command for an Automation Module.
- An Update Status Instruction consists of an Element as discussed earlier. The work station controller will update the world model with this Status.
- Move Status Instruction consists of two Variables.

In order to execute a Move Status Instruction, the work station controller will update the world model, by assigning to the status element with the second Variable, the value associated with the first Variable, and assign the value 'noResource' to the first Variable.

Assume, for example, that the world model contains the following two status elements:

location1	Productx
location2	NoProduct

After the execution of the Move Status Instruction,

location1	location 2

the world model contains the following two Status elements:

location1	noResource
location2	Productx

Execution of a move status instruction

The Move Status Instruction is used to update the world model if products have been transported from one place in the work station controller's domain to another.

When a Recipe Line is processed, all Recipe Commands of its Set of Recipe Instructions are executed. The order of execution is arbitrary.

4. Next Step Number identifies the Step Number of a Recipe Line that should be executed next. There is one special Step number, 'exit' which means that there is no next Recipe Line that should be executed.

The following scheme shows the sequence of activities that take place when a recipe is processed.

```
                          |
                 1: select Recipe
                          |
               2: actualise Recipe
                          |
            3: Step Number:= init
                          |
                          v
+------------------->|
|                         |
|         4: select Recipe Line,
|  depending on step and status feedback
|                         |
|            5:  process Recipe Line
|                         |
|     6: Step Number := Next Step Number
|                         |
|                         v               yes
|         <Step Number = exit> -------------> exit
|                         v
|                         | no
|                         |
+--------------------+
```

1. The work station controller selects a Recipe that has an identifier that matches the Operation Description of an Operation Instruction that should be executed.
The possible forms of matches are:

1. The Operation Description and the Recipe Identity are identical.

2. The Product Identifier of the Operation Description lies within the range of Product Identifiers specified in the Recipe Identity.

2. The Recipe has to be actualised.
The Recipe may contain variables for which actual values should be substituted during processing. These variables concern Product Identifiers. Assume, for example, a Recipe Identity of a Recipe for a dispatch operation. In the Recipe, the product that is dispatched is referred to by a variable 'x'. Assume that this Recipe Identity matches by an Operation Description 'Dispatch product1'. Now, the actualised recipe will contain the value 'product1' instead of the variable 'x'.

3. The Step Number of the first Recipe Line that is processed in 'init'.

4. A Recipe Line has to be selected for processing. A Recipe Line can only be processed if its Step Number has the right value and if all its Conditions are satisfied. The work station controller will inspect its world model to decide whether or not all the Conditions of a particular Recipe Line are satisfied.
If there is more than one Recipe Line with the same Step Number, the work station controller will select one for evaulation in an arbitrary order until it finds one Recipe Line that is suitable. If no Recipe Line is eligible for execution, the selection will be repeated (changing status conditions may change the suitability of a recipe Line for processing).

5. The selected Recipe Line is Processed. This means that the Recipe Instructions contained in the selected Recipe Line have been executed. A Recipe Line may contain more than one Recipe Instruction. These are executed in an arbitrary order. There are three types of Recipe Instructions, viz.

- Send Message Instructions
- Update Status Instructions
- Move Status Instructions

6. The Step Number of the next Recipe Line to be processed is the Next Step Number of the most recently processed Recipe Line. If this number is 'exit', the processing of the Recipe has finished. If not, the next Recipe Line has to be processed.

Simultaneous Use Of Resources By More Than One Recipe

Development of a recipe can be considered as the development of a program for a work station controller. The variables referred to in the set of conditions of a recipe line are global variables accessible for all recipes that are executed, as they are in the world model.

The work station controller will execute only one recipe line at a time and the next recipe line can only be started if all instructions of a previous recipe line have been executed.

It may occur that more than one recipe is executed simultaneously and that it is necessary to issue commands for a single automation module, space, tool or product. These commands can easily pose conflicting claims on it. For example, one recipe requires an automation module to move from A to B whereas another recipe requires an automation module to move from A to C.

Such conflicting claims can be avoided if the recipes are programmed correctly. The global character of a status variable and the one-by-one execution of recipe lines can be used to this end.

The following mechanism is used. A recipe may request the updating of status in the world model. Status is assumed to consist of a Variable that describes the resources and a Value that describes the Identity of a recipe that controls the resource. By updating the Value of such status types, the rights to control the resource can be tranferred.

Assume, for example, the variable 'criticalAM' and a recipe with identity 'criticalAMUser'. The following lines of a recipe guarantee that the automation module 'criticalAM' is controlled by the recipe 'criticalAMUser' only.

n	criticalAM = noRecipe	US(criticalAM,criticalAMUser)
n+1		
n+1		
:		:
:		
m		US(criticalAM,noRecipe)

The standard value 'noRecipe' indicates that no recipe controls the 'criticalAM'. By executing the Update Status Instruction (US), it is indicated that the recipe 'criticalAMUser' now controls the automation module.

SET OF OPERATION DESCRIPTIONS

A work station cannot execute all possible operations. For example, due to a certain physical implementation, it cannot execute operatios that require high temperatures. As a consequence, the work station controller may reject the execution of some operations.

During start-up, the work station controller should be fed with Descriptions of the operations it can execute. There are several ways to determine whether a given operation can be executed.

SET OF AUTONOMOUS OPERATIONS

Often a work station controller will be able to execute some operations directly upon becoming active, that is, without an explicit command from a host. An example is the handling of materials and tools in the domain of a work station in order to position these at optimal places.

At start-up, the workstation controller should be fed with the Operation Descriptions of these operations. Apart from the types of Operation Descriptions mentioned in the previous section, an Operation Description can be used that only defines the identifier of an Operation.

FORMULAE

A Formula is a prescription of how a work station controller should execute a received status element, apart from just updating the world model with it. A Formula consists of a Formula Variable and one or more

Formula Lines, as follows:

```
|Formula Variable
|
         :
         :
         :
         :
|Formula Line
|Formula Line
|Formula Line
|Formula Line
|
```

A Formula Variable is a Status Variable, its Formula will be processed if a status indication is received that contains this variable.

A Formula Line contains two parts:

1. Set of Conditions and
2. Formula Instructions

The meaning of these are identical to the meanings of Set of Conditions and Set of (Recipe) Instructions of a Recipe. However, the Formula Instructions do not contain messages to be sent to the automation modules or transport system. They can contain messages to the host.

Conditions	Instructions

A Formula is processed as follows:

1. The work station controller selects a Formula that has a Formula variable matching the received Status Variable.
2. Each formula is evaluated. The instructions of a formula line can only be processed if all Conditions are satisfied. The work station controller will inspect its world model to decide whether or not all Conditions of a particular Formula Line are satisfied. If this is the case :

- The selected Formula Line is Processed. As with the instructions in a Recipe Line, this means that the Instructions contained in the selected Formula Line are executed.

Caption listing

Figure 1: 60 manager; 62 policy; 64, 66 host; 68 when; 70 work station controller; 72 development system; 74 status aggregation; 76 how; 78 processing/automation module; 80 transport module; 82 what; 84 product.

Figure 2: 20 host; 21 manager module; 22 work station controller module; 23 status management module; 25 model description module; 72 interaction module; 24, 26, 28 automation modules; 30, 32 transport system modules; 34 product.

Figure 3: 100 host level: procedure product from raw material; 102 work station controller level: unit product transformation; 104 automation module control level: modifies physical entities; 110, 112, 114 status aggregation; 120, 122, 124 world model; 130, 132, 134 command decomposition.

Claims

1. A work station controller module for controlling a work station and comprising processor means interconnected to storage means for on the basis of predetermined operation-linked process data and under

control of higher level commands and lower level status signals producing lower level commands and higher level status signals, said work station controller module further comprising:
- first bidirectional communication means for receiving said higher level commands from and transmitting said higher level status signals to a host device;
- second bidirectional communication means for receiving said lower level status signals from and transmitting said lower level commands to one or more product processing automation modules;
- third bidirectional communication means for transmitting product exchange requests and receiving product exchange acknowledgements with respect to any access point the work station has in common with a product transfer system or with at least one other work station;
first receiving means for receiving operation names and process data from a product development system; second receiving means for receiving status aggregation rules for aggregating said lower level status signals to said higher level status signals and command decomposition rules for decomposing said higher level commands to said lower level commands from a product preparation system.

2. A work station controller module as claimed in Claim 1, wherein said work station controller attends to said host device as first source for therefrom receiving timing signals and quantization signals for work station product transform and attends to a second source that may be different from said first source for therefrom receiving modality signals for modality controlling said work station product transform.

3. A workstation controller module as claimed in Claim 1 or 2, comprising four mutually cooperating process modules, as follows:
- an interaction module H for receiving and updating said higher level commands, for generating said lower level commands, and receiving said lower level status signalling any of initiation/non-initiation/termination/non-termination of execution of any of said second lower level commands, for managing said operation names/process data, and for activating said third communication means;
- a model description module M for managing model information of a world model, inclusive of parameter data of said automation module, any assigned product and any assigned tool;
- a status management module G for receiving and aggregating said lower level status signals to said highe level status signals and for checking any actual status to said world model for upon incompatibility generating an alarm signal for said interaction module, and
- a manager module A for specifying
o logical addresses for said host device;
o allowable physical locations for product/tools;
o alarm conditions for specifying said incompatibility;
o and for presenting application-dependent information to said interaction module, said model description module, and said status management module.

4. A work station controller module as claimed in Claim 1, and comprising strategy defining means for defining a strategy of executing a higher level command as represented by associated lower level commands without this higher level command defining such strategy.

5. A work station controller module as claimed in Claim 1, and comprising checking means for checking any higher level command received versus an associated recipe stored locally and upon the result of said checking specifying any associated lower level command.

6. A work station controller module as claimed in Claim 1 wherein said higher level commands comprise any of the following: suspend, cancel, resume.

7. A work station controller module as claimed in Claim 6, and comprising specifying means for numerically specifying a serially repetitively executable series of operations to be suspended on/or cancelled.

8. A work station controller module having multiprocessing facilities for in each of a respective set of processes controlling a respectively associated automation module domain.

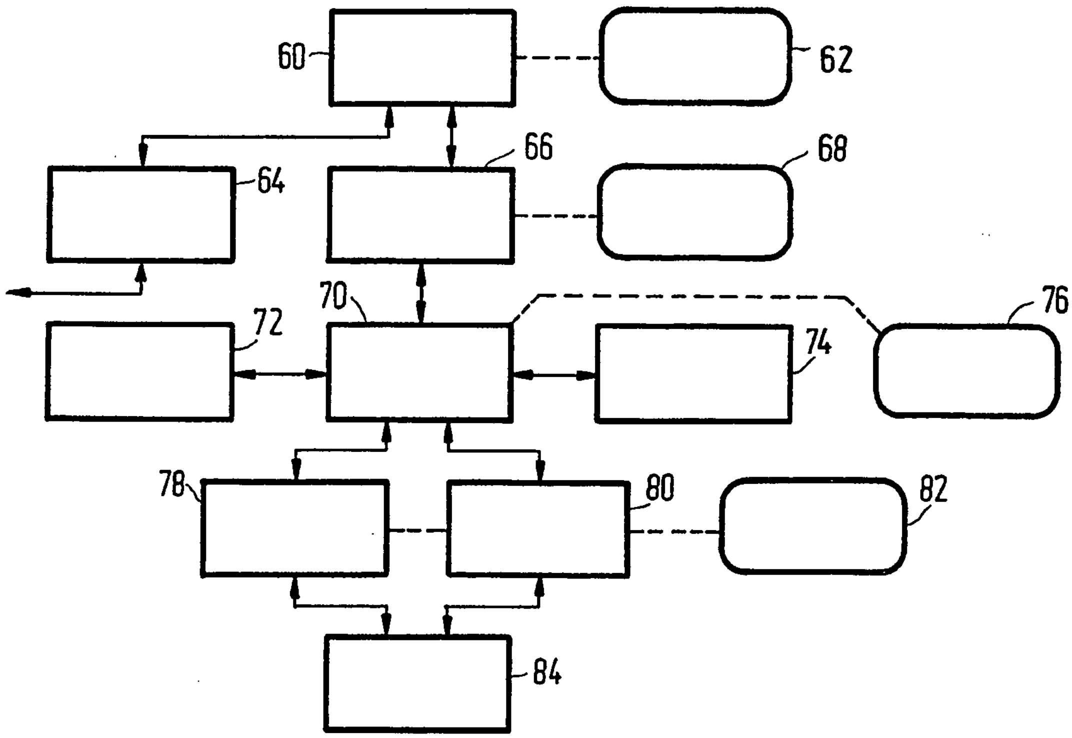
60
62
64
66
68
70
72
74
76
78
80
82
84

FIG.1

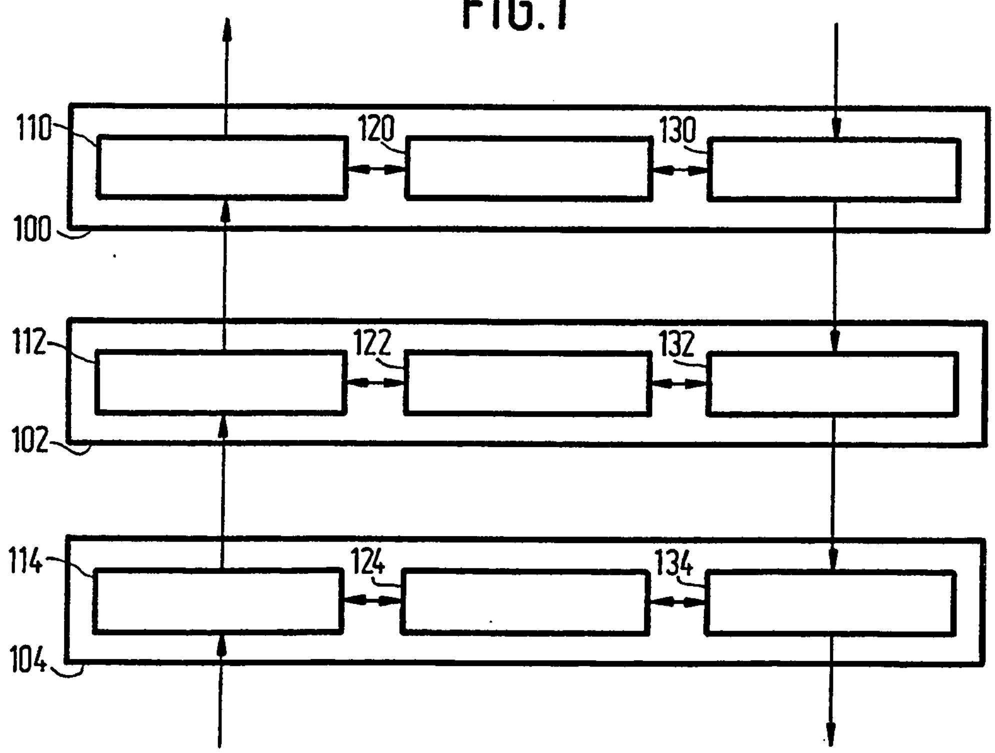
110
120
130
100
112
122
132
102
114
124
134
104

FIG.3

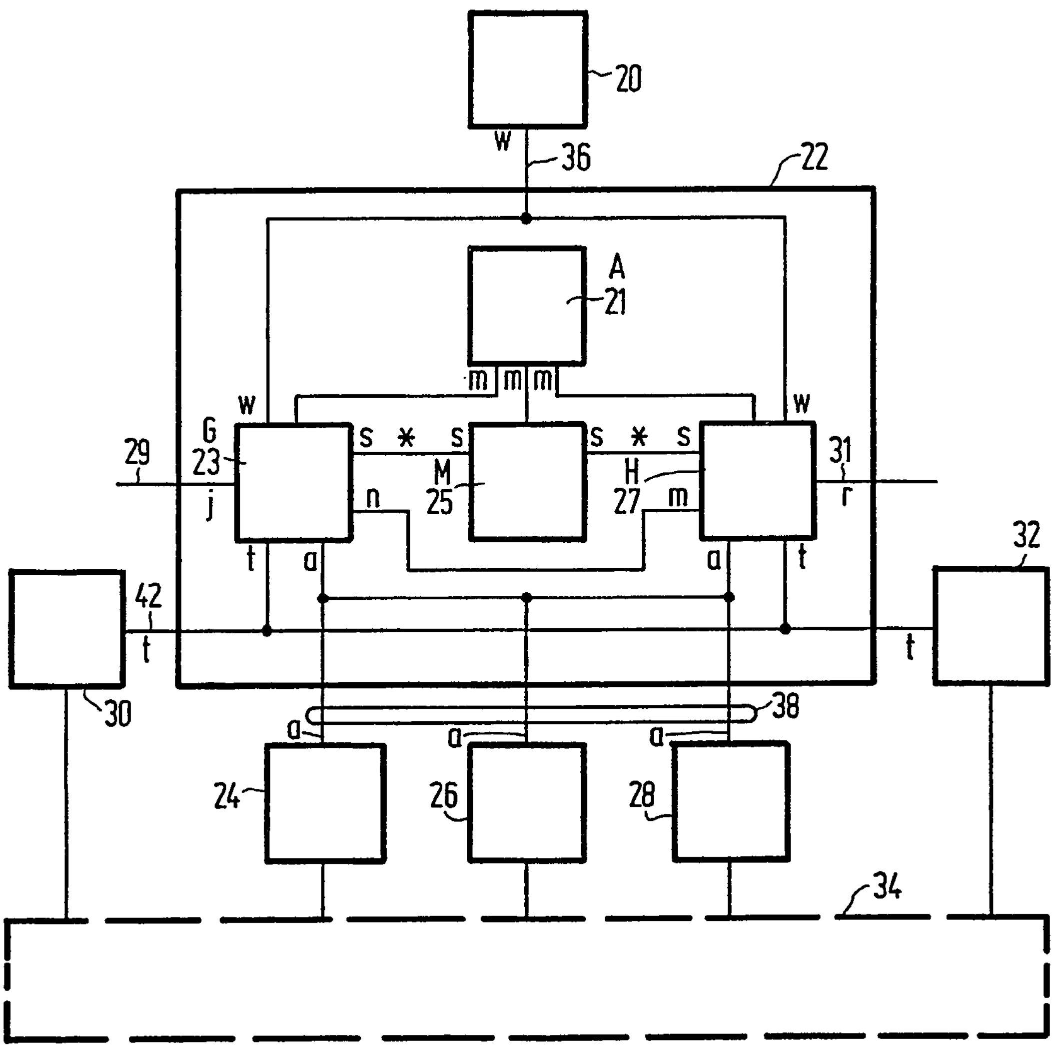
20
w
36
22
A
21
m
m
m
w
w
G
23
s
*
s
s
*
s
29
M
25
H
27
31
j
n
m
r
t
a
a
t
32
42
t
t
30
38
a
a
a
24
26
28
34

FIG.2

European Patent Office

EUROPEAN SEARCH REPORT

Application Number

EP 89 20 1248

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
X	EP-A-0 266 784 (ALLEN-BRADLEY CO.) * page 3, lines 3-39; page 8, line 17 - page 9, line 4; page 11, line 13 - page 12, line 35; page 13, lines 1-19; claims 1-13; figures 3,4,7,8 * ---	1,4,8	G 05 B 19/417
A	EP-A-0 162 670 (BRITISH AEROSPACE) * page 3, line 1 - page 5, line 17; page 5, line 24 - page 17, line 14; page 23, line 18 - page 24, line 23; claims 1,2; figures 1,2 *	1,2,4-6	
X	---	8	
A	GB-A-2 049 243 (FORNEY ENGINEERING) * abstract; page 1, line 57, - page 3, line 8; page 12, lines 25-44; claims 1,3-10; figure 1 * ---	1,5	
A	US-A-4 698 629 (K. MORI et al.) * column 1, lines 29-56; column 2, line 6 - column 4, line 2; claims 1,2; figures 1-7 * ---	1,2,4,5	TECHNICAL FIELDS SEARCHED (Int. Cl.5)
A	EP-A-0 100 684 (FANUC) * page 2, lines 24 - page 4, line 1; page 4, line 21 - page 9, line 8; page 14, line 16 - page 20, line 12; claims 1-4; figures 1-5 * -----	1,5,7	G 05 B 19/00 G 06 F 15/00

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
BERLIN	09-01-1990	BEITNER M.J.J.B.

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

TASK PLANNING FOR FLEXIBLE AND AGILE MANUFACTURING SYSTEMS

João Rocha①, Carlos Ramos①②

①
Departamento de Engenharia Informática
Instituto Superior Engenharia do Porto
Rua S. Tomé
4200 PORTO
PORTUGAL
Phone: +351 2 830 09 22
FAX: +351 2 82 11 59

②
Departamento de Engenharia Electrotécnica
e Computadores
Faculdade de Engenharia da Universidade do Porto
R. Bragas
4099 PORTO CODEX
PORTUGAL
Email: csr@garfield.fe.up.pt

ABSTRACT

This paper deals with Task and Execution Planning for Manufacturing Systems.

We describe a system named TPMS[1] (Task Planning for Manufacturing Systems). The final goal of this system is to automatically generate programs or sets of commands for robots, AGV's, numerical control machines and other components of the system without the need of spending much time programming the Flexible Manufacturing System (FMS) components.

The sequence of operations to be handled by the Manufacturing System depends on the constraints for the task. Three types of constraints are defined and explained: processing constraints; feasibility constraints and geometric constraints. The symbolic plan for the task is represented in the form of a precedence graph. High level operations can be converted into programs or sets of instructions to control the Manufacturing System's equipment.

The main contributions of TPMS are the following: to put together symbolic planning and execution planning; it is being developed to work for complete Manufacturing Systems (and not for controlling only one robot); different industrial operations (assembly, welding, drilling,...) can be considered at the same time.

KEY-WORDS: FMS, Task Planning, Execution Planning, Constraints

1 This work is partially supported by FLAD ("Fundação Luso-Americana para o Desenvolvimento") under project 443/93 - SFF

1. INTRODUCTION

On the last years, we have observed fundamental changes on markets for manufacturing products. Today, Manufacturing Systems markets are more global, complex and competitive than before.

An important challenge is the reduction of the products life-cycles and the reduction of the portion dimension. This implies a need to frequently redesign products and reprogram Manufacturing Systems. However, Manufacturing Systems reprogramming is a complex task involving a considerable set-up time. Most of this time is spent in defining how the new product will be manufactured and in programming the different machines of the Manufacturing System (e.g. CNC, robots, AGV's, conveyors, ...).

The increase of system's flexibility is very important to achieve efficiency and productivity improvements. Now the emphasis is on versatility and intelligence leading to ideas such as Intelligent Design and Automatic Planning and Programming.

This paper deals with Task and Execution Planning for Manufacturing Systems. Task and Execution Planning involve two phases. In the first it is generated a symbolic plan for the Manufacturing Task. In the second, the symbolic operations of the high level plan are converted in sets of programs, instructions and commands to control the Manufacturing System's equipment. For this purpose we are developing a system named TPMS [1,2] (Task Planning for Manufacturing Systems) that will be presented in this paper.

DOCID: <XP__10141834A__I_>

2. STATE OF THE ART AND BACKGROUND

Some systems have been developed to deal with Automatic Task Planning and Execution. Most of them only take into account high level symbolic planning without machine or robot programming.

Bourjault [3] has described an interactive system for assembly sequence generation of a product. It is created a liaison graph representing parts connections. However, this system has not inference capabilities concerning geometric knowledge about the assembly. This knowledge is furnished by the user who answers to questions about the sequence of operations.

De Fazio [4] has improved this methodology reducing the number of questions to the user. In this system the user writes logic formulas representing the sequence of operations. The user needs to be familiar with this syntax and, sometimes, it is difficult to map from assemblies to the formulas. It is also created a liaison graph.

Homem de Mello [5] uses AND/OR graphs to represent assemblies. With this kind of graphs, the number of nodes is drastically reduced, namely for tight coupled assemblies. It is possible to generate all feasible assemblies from the AND/OR graph. However, the combinational explosion problem may happen here. There are some feasibility tests (geometric feasibility, mechanical feasibility and stability feasibility) that are applied to test if a specific assembly sequence is really feasible.

Wilson [6] has improved the efficiency of plan generation from AND/OR graphs using some kind of geometric reasoning. Now, only feasible sequences of operations are generated.

Sukhan Lee [7] has developed COPLANNER, a distributed and co-operative approach to Planning with the Blackboard paradigm. Concerning Planning, this system uses the Backward Assembly Planning (BAP). Sukhan Lee illustrates some examples in which a plan cannot be achieved by the inversion of the plan generated backwards. However, BAP is able to detect these problems, achieving feasible solutions.

The five systems described above did not consider the existence of a manufacturing system for performing the assembly task. The system from Huang and Lee [8,9] accepts as input a product description from a CAD system and generates an assembly plan, subjected to some resource restrictions inherent to a manufacturing cell. There is a "world" database representing the components, relations among them, industrial operations, machines and tools from manufacturing cell. It is generated a "world" simulated model for the various stages that exist during the assembly. There is a knowledge acquisition mechanism with three modules: precedence between operations acquisition module, fixing operations specification's module and tools selection module. The plan generator makes inferences on a set of production rules like "IF <pre-condition> THEN <actions>". Actions correspond to a Delete List and a Junction List. The plan is obtained by inversion of the dismount sequence.

All the systems described above did not consider the Execution Planning, i.e., how to program machines and/or robots to perform the generated high level symbolic plan. However, some systems dealt with this question in the area of Robotics. The group of Lozano-Perez [10] has implemented HANDEY, a system for performing grasping and regrasping operations for pick and place problems. However, this system has some limitations concerning the reasoning about action. Hutchinson [11] has presented another interesting approach to this problem in SPAR system. SPAR uses a three-level hierarchy with operational, geometric and uncertainty-reduction goals. Unfortunately, collision avoidance is not considered in this work.

TPMS takes advantage of CIARC (Cooperative Intelligent Assembly Robotic System). CIARC has been implemented for Task Planning and Execution of Manipulation and Assembly Tasks using one robot arm programmed by a textual language [12,13,14]. The conversion of symbolic plan operations is situation dependent and some aspects, such as collision avoidance and intelligent execution, were considered in CIARC system.

While including the ideas of Intelligence and Flexibility, CIARC was developed for Assembly and Manipulation Robotics, controlling the operation of only one robot.

TPMS will be the extension of CIARC to complete Manufacturing Systems. The final goal is to automatically generate programs or sets of commands for robots, AGV's, numerical control machines and other components of the system without the need of spending much time programming the several Manufacturing System components.

In the current manufacturing cells the operator is the one who defines the sequence of operations to be carried out and the way these ones will be performed by the different components of the system (robots, numerical control machines, AGV's, ...). The purpose of the research work addressed in this paper is to develop a software to replace (or help) the manufacturing cell

SDOCID: <XP_10141834A_I_>

programmer, i.e., to automatically obtain a plan to the industrial task and to manage the control code (program, instructions) of each machine and robot.

This is an important contribution to reduce the time spent for task activation, since it helps the programmers in developing programs for the manufacturing of new products. This work points out in the direction of the new trends of Manufacturing Systems: intelligence, agility and flexibility [15,16].

3. HIGH LEVEL TASK PLANNING

Task and Execution Planning involve two phases: first a high-level plan composed of symbolic operations is automatically generated and represented (e.g. put A on B, weld C on D, insert E in F); then the programs for controlling the several machines of the Manufacturing System are generated (e.g., a program for controlling a robot to insert an object into a hole of another object).

The high-level planning makes possible to generate a plan considering all processing, symbolic and geometric constraints. It is also possible to generate not only one, but several plans that could be dynamically activated according to the current situation.

Let us consider some basic definitions about "High Level Symbolic Planning". Planning is a research area in which it is tried to answer to the following question: "How to solve a problem", or in other words "How to achieve a goal or a set of goals". To solve this problem we need to go from an initial state to a desired final state.

Operators are the elements that can be used to change the state of the "world". Operators can be seen as general procedures in which the variables have not values. For example, we can consider the following operators:

- pickup(Ob)
- insert(Ob1,Ob2)
- put_on(Ob,LOb)
- drill(Ob)
- weld(LOb)
- mill(Ob)
- transport_input_conveyor(Ob)

Operations are instantiations of operators; an operation corresponds to the instatiation of all variables in the arguments of the operator. If our basic objects or parts are in the set [a, b, c, d, e, f] then the following operations are valid:

- pickup(a)
- insert(a,b)
- put_on(c,[a,d])
- drill(e)
- weld([e,a])
- mill(e)
- transport_input_conveyor(f)

Once an operation is applied the state of the parts' world changes (e.g. figure 1).

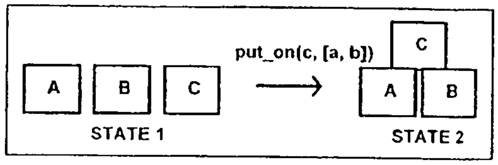

figure 1

Some operators are able to transform parts or change their shapes (e.g. drilling or milling) and other are able to join parts (e.g. welding). This means that after an operation we may have a transformation of the parts handled by the operation. To represent an operation and the new part that results from the operation we will use the following syntax:

operation => result

Let us see some examples using this format:

drill(a) => a_d	means that part a was drilled giving origin to a new part named a_d
weld([a,b]) => ab	means that parts a and b are welded giving origin to a new part named ab

The general goal we want to achieve is composed by a set of subgoals. We will use operators to achieve these subgoals. The sequence of operations for the task is the symbolic plan for the task.

However, the subgoals cannot be always achieved in any order. Often, the accomplishments of some goals are restricted by constraints. For Manufacturing Tasks the main constraints are the process constraints, feasibility constraints and geometric constraints. The constraints imply precedence relationships to guarantee that operations will be executed in the correct order. The syntax we will use will be the following:

operation1 *before* operation2

SDOCID: <XP_10141834A_I_>

As examples of precedence relationships between operations we can have:

- drill(a) before weld([a,b])
- insert(a,b) before insert(c,b)

It is important to acquire the knowledge about these constraints. The knowledge about the feasibility and geometrical constraints can be automatically acquired with some geometric reasoning capabilities [12,13,14].

As an example of feasibility constraints' detection we can consider the case in which 2 subgoals: one to drill object b and another to insert object a in the hole of b. The second can be accomplished only after the accomplishment of the first.

Feasibility constraints are detected by the verification of a set of feasibility constraint rules. An example of a rule of this type may be the following:

For insert(Ob, Ob1)

- Ob must be available
- Ob1 must be available
- the hole of Ob1 must be performed.

Geometric Constraints are obtained by the analysis of the geometry of the operation. Collisions are a complex problem to deal with in Manufacturing Tasks. The geometric constraints related with the collision analysis can be automatically obtained as it happens in the CIARC system where two kinds of constraints related with collisions are automatically acquired: constraints to avoid collisions between a robot and the objects and constraints to avoid collisions between objects being handled and other objects [12,13,14].

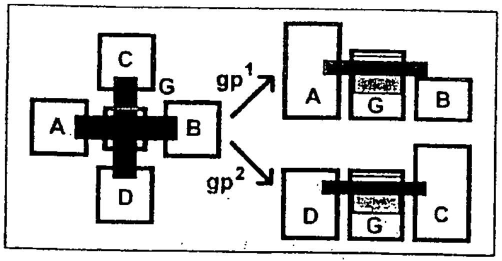

figure 2

For the example of figure 2 the black rectangle of the top view represents the area swept by the 2 jaw mechanical gripper during grasping or ungrasping of object G. The grey rectangles of the lateral view represent the accessibility zone (where the robot gripper can grasp the object G).

For this example we have the following geometric constraints for the gripper-object collision analysis:

- object G is obstructed by object A in one grasping pose (gp1)

- object G is obstructed by objects D and C in the other pose (gp2)

This means that we will have the following constraints:

{put_on(g,[pallet]) before put_on(a,[pallet])}
or
{ {put_on(g,[pallet]) before put_on(d,[pallet])}
and
{put_on(g,[pallet]) before put_on(c,[pallet])} }

3.1. An example

Let us consider an example for assembling a support for 3 cylinders. There are 5 parts: 3 cylinders (*C1, C2, C3*) with different heights; the base *B* and the lateral wall *W* (figure 3).

The parts *B* and *W* need to be processed by a drilling machine (3 holes in *B* and 1 hole in *W*). These 2 parts will be welded, giving origin to a new part. The cylinders *C1, C2* and *C3* need to be inserted in the base.

By default, we do not know the order by which the parts will be processed and assembled. The sequence of operations will be obtained by the feasibility and geometric constraints between parts and tools or grippers. There are also some processing constraints.

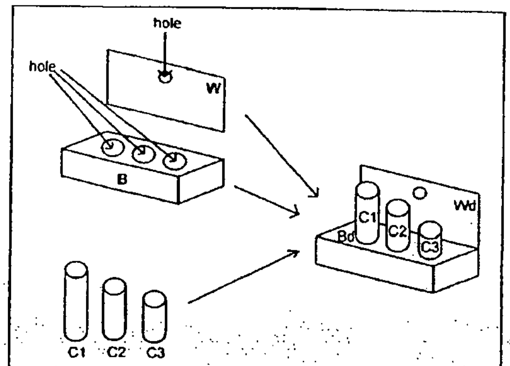

figure 3

The following equipment composes the manufacturing cell for this example:

SDOCID: <XP__10141834A__I_>

- 1 drilling machine
- 1 welding robot
- 1 assembly robot (for insertions, loading and unloading)
- 1 input conveyor and 1 output conveyor

The operations to be considered are the following:

- insert *C1* in the hole of *B* → *insert* (*C1*, *B*)
- insert *C2* in the hole of *B* → *insert* (*C2*, *B*)
- insert *C3* in the hole of *B* → *insert* (*C3*, *B*)
- drill 1 hole in *W* → *drill*(*W*)
- drill 3 holes in *B* → *drill*(*B*)
- weld *B* and *W* → *weld*([*B*, *W*])
- transport B on the input conveyor → *transport_input_conveyor*(*B*)
- transport W on the input conveyor → *transport_input_conveyor*(*W*)
- transport C1 on the input conveyor → *transport_input_conveyor*(*C1*)
- transport C2 on the input conveyor → *transport_input_conveyor*(*C2*)
- transport C3 on the input conveyor → *transport_input_conveyor*(*C3*).
- transport the manufactured object on the output conveyor → *transport_ouput_conveyor*(*Obm*)
- pick up B from the input conveyor → *pickup*(*B*)
- pick up W from the input conveyor → *pickup*(*W*)
- pick up C1 from the input conveyor → *pickup*(*C1*)
- pick up C2 from the input conveyor → *pickup*(*C2*)
- pick up C3 from the input conveyor → *pickup*(*C3*)
- put the manufactured object at the output conveyor → *putdown(Obm)*
- load drilling machine with B → *load_drilling_machine*(*B*)
- load drilling machine with W → *load_drilling_machine*(*W*)
- unload drilling machine with B → *unload_drilling_machine*(*B*)
- unload drilling machine with W → *unload_drilling_machine*(*W*)
- load welding robot with B → *load_welding_robot*(*B*)
- load welding robot with W → *load_welding_robot*(*W*)
- unload welding robot with B welded with W → *unload_welding_robot*(*BW*)
- load assembly robot with B and W → *load_assembly_robot*(*BW*)
- unload assembly robot with the assembled object → *unload_assembly_robot*(*Obm*)

However, we do not know in which order they will be carried out. To know this, we will generate all feasibility, geometric and processing constraints. They are the following:

<u>processing constraints:</u>

- for this class of assemblies we must drill before to weld. This means:
  drill(*W*) before *weld*([*B*, *W*])
  and
  drill(B) before weld([B, W])

<u>feasibility constraints:</u>

- if some parts are to be inserted, then the holes must be performed. This means:
  drill(*B*) before *insert*(*C1*, *B*)
  drill(*B*) before *insert*(*C2*, *B*)
  drill(*B*) before *insert*(*C3*, *B*)
- transport an object before to pick up it
  transport_input_conveyor(X) before pickup(X)
  X=B or X=W or X=C1 or X=C2 or X=C3
- pickup an object before loading machines and robots
  pickup(X) before load_drilling_machine(X)
  X=B or X=W
  pickup(X) before load_welding_robot(X)
  X=B or X=W
- load machines and robots before processing
  load_drilling_machine(X) before drill(X)
  X=B or X=W
  load_welding_robot(B) before weld(BW)
  load_welding_robot(W) before weld(BW)
  load_assembly_robot(BW) before insert(X,BW)
  X=C1 or X=C2 or X=C3
- process before unloading
  drill(X) before unload_drilling machine(X)
  X=B or X=W
  weld(BW) before unload_welding_robot(BW)
- pickup objects before assembly or manipulation operations
  pickup(X) before insert(X,BW)
  X=C1 or X=C2 or X=C3
- put down objects before to transport
  putdown(Obm) before
  transport_output_conveyor(Obm)

<u>geometric constraints:</u>

- concerning welding

  if we consider *C1*, *C2*, and *C3* already inserted before welding we will see that collisions will occur

DOCID: <XP__10141834A__I_>

between the welding tool and these cylinders. This means:

weld([B, W]) before *insert(C1, B)*
weld([B, W]) before *insert(C2, B)*
weld([B, W]) before *insert(C3, B)*

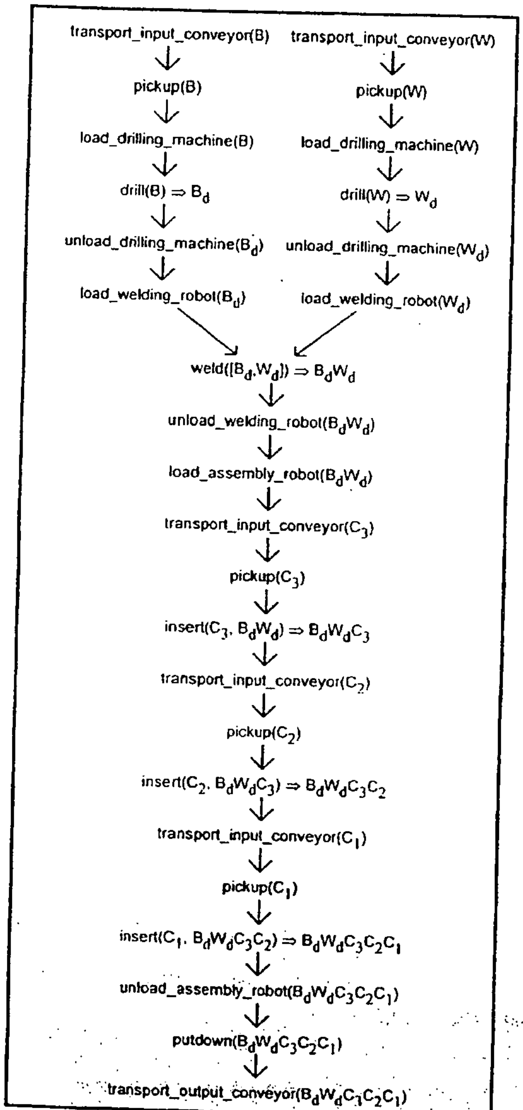

figure 4

• concerning insertions

since the robot uses a 2 jaw mechanical gripper, the only possibility to avoid collision is to consider the jaws parallel to *W*. In this case notice that if *C3* is inserted after *C2*, a collision between the jaws and *C2* will occur. Also, if *C2* is inserted after *C1*, a collision between the jaws and *C1* will occur. Thus:

insert(C2, B) before insert(C1, B)
and
insert(C3, B) before insert(C2, B)

Figure 4 shows the precedence graph that is automatically generated by our planning methodology.

In this precedence graph some operations (drill, weld, insert) have the special symbol $\Rightarrow$, at the right of this symbol we have the result of the operation.

Notice that the two parallel branches of the precedence graph are non-preemptible sets of operations. This happens because both operations are related with the same resource (one drilling machine). With 2 drilling machines we could have interleaving between operations of the first and second sets (e.g. transport and pick up B and load one drilling machine; transport and pickup W and load the other drilling machine; drill B; drill W ...).

4. EXECUTION PLANNING

After the elaboration of the high level plan for the task, it is necessary to plan how each specific operation will be executed (e.g. - how to program the drilling machine, welding robot, assembly robot and conveyors).

Here we will illustrate the code generated to program the robot for the three insertions of objects C1, C2 and C3. For this purpose we will consider figure 5. In this figure it is shown that the minimum and maximum heights at which it is possible to grasp objects *C1*, *C2* and *C3*.

We will consider that the height of the bottom of the holes of *B* is 10 mm and that the top of the holes is at height 25 mm.

Let us consider the following values:

(X_i, Y_i, 0) - position of the bottom cylinders on the input conveyor, all of them will stop at the same position

(X_1, Y_1, 10) - position of the bottom of *C1* after the insertion in the hole of *B*.

SDOCID: <XP__10141834A__I_>

$(X_2, Y_2, 10)$ - position of the bottom of *C2* after the insertion in the hole of *B*

$(X_3, Y_3, 10)$ - position of the bottom of *C3* after the insertion in the hole of *B*

$ALFA_i$ - orientation of the gripper to grasp the cylinders

$ALFA_f$ - orientation of the gripper to ungrasp the cylinders

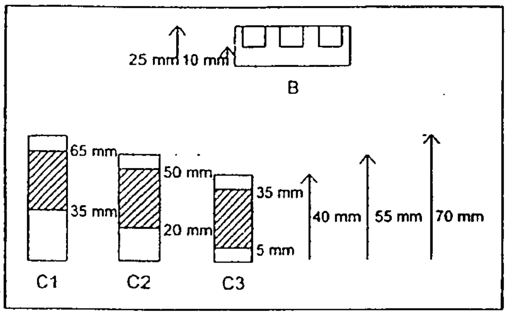

figure 5

Considering *C3* alone, we can grasp it in the interval [5 mm, 35 mm]. However, the hole avoids the possibility to consider the interval between 5 mm and 15 mm. Thus, the real height interval to grasp *C3* is [15 mm, 35 mm]. Considering 3 mm of tolerance, the interval is [18 mm, 35 mm].

Therefore, the code to insert *C3* in *B* can be the following:

MOVE_ROBOT_G_TO(X_i, Y_i, H_{safe}, $ALFA_i$);
MOVE_ROBOT_G_TO(X_i, Y_i, 43, $ALFA_i$);
(* 43 = 40 + tolerance to avoid collision with the top of C_3*)
OPEN_GRIPPER;
MOVE_ROBOT_G_TO(X_i, Y_i, 18, $ALFA_i$);
(* 18 = 15 + tolerance*)
CLOSE_GRIPPER;
MOVE_ROBOT_G_TO(X_i, Y_i, H_{safe}, $ALFA_i$);
MOVE_ROBOT_G_TO(X_3, Y_3, H_{safe}, $ALFA_f$);
MOVE_ROBOT_G_TO(X_3, Y_3, 43, $ALFA_f$);
(* 43 = 25 + 18*)
MOVE_ROBOT_G_TO(X_3, Y_3, 28, $ALFA_f$);
(* 28 = 10 + 18*)
OPEN_GRIPPER;
MOVE_ROBOT_G_TO(X_3, Y_3, H_{safe}, $ALFA_f$);
CLOSE_GRIPPER;

where H_{safe} is a height sufficiently high to avoid collisions and 3 procedures (MOVE_ROBOT_G_TO; OPEN_GRIPPER and CLOSE_GRIPPER) are used for manipulations.

Considering *C2* alone we can grasp it in the interval [20 mm, 50 mm]. But, due to the constraint imposed by *C3* during ungrasping, we cannot grasp for heights below 40 mm. Thus, the real interval is [43 mm, 50 mm], with 3 mm of tolerance.

In this case the code to insert *C2* in *B* is:

MOVE_ROBOT_G_TO(X_i, Y_i, H_{safe}, $ALFA_i$);
MOVE_ROBOT_G_TO(X_i, Y_i, 58, $ALFA_i$);
(* 58 = 55 + tolerance*)
OPEN_GRIPPER;
MOVE_ROBOT_G_TO(X_i, Y_i, 43, $ALFA_i$);
CLOSE_GRIPPER;
MOVE_ROBOT_G_TO(X_i, Y_i, H_{safe}, $ALFA_i$);
MOVE_ROBOT_G_TO(X_2, Y_2, H_{safe}, $ALFA_f$);
MOVE_ROBOT_G_TO(X_2, Y_2, 68, $ALFA_f$);
MOVE_ROBOT_G_TO(X_2, Y_2, 53, $ALFA_f$);
OPEN_GRIPPER;
MOVE_ROBOT_G_TO(X_2, Y_2, H_{safe}, $ALFA_f$);
CLOSE_GRIPPER;

Now, considering *C1* alone the interval is [35 mm, 65 mm]. Due to the constraints imposed by *C2* during ungrasping, this interval is reduced to [58 mm, 65 mm], having 3 mm of tolerance.

Thus, the code to insert *C1* in *B* is:

MOVE_ROBOT_G_TO(X_i, Y_i, H_{safe}, $ALFA_i$);
MOVE_ROBOT_G_TO(X_i, Y_i, 73, $ALFA_i$);
(* 73 = 70 + tolerance*)
OPEN_GRIPPER;
MOVE_ROBOT_G_TO(X_i, Y_i, 58, $ALFA_i$);
CLOSE_GRIPPER;
MOVE_ROBOT_G_TO(X_i, Y_i, H_{safe}, $ALFA_i$);
MOVE_ROBOT_G_TO(X_1, Y_1, H_{safe}, $ALFA_f$);
MOVE_ROBOT_G_TO(X_1, Y_1, 83, $ALFA_f$);
(* 83 = 58 + 25*)
MOVE_ROBOT_G_TO(X_1, Y_1, 68, $ALFA_f$);
(* 68 = 58 + 10*)
OPEN_GRIPPER;
MOVE_ROBOT_G_TO(X_1, Y_1, H_{safe}, $ALFA_f$);
CLOSE_GRIPPER;

Notice that in all three insertions the position where we grasp each cylinder in the pickup(Ci) operation depends on the position where we can ungrasp it in the operation insert(Ci,BW_).

5. CONCLUSIONS AND FUTURE WORK

This paper has described and illustrated TPMS (Task Planning for Manufacturing Systems). Some definitions related with symbolic task planning have

SDOCID: <XP__10141834A__I_>

been presented and precedence relationships are obtained from feasibility constraints, geometric constraints and process constraints. The main contributions of TPMS are the following:

- to put together symbolic planning and execution planning (as in CIARC)
- to be prepared to work for Manufacturing Systems (and not for controlling only one robot).
- to be prepared to deal with different industrial operations and not only for assembly

The future trends of TPMS will lead to some new performances, namely:

- the integration with a scheduling policy
- the improvement of the automatic conversion of the symbolic plan into programs to control machines, robots and other equipment
- the interaction with an Intelligent Design System.

Concerning the integration with scheduling, the main idea is to generate a structure for representing symbolic plans and programs for controlling the FMS equipment in such a way that this structure could be dynamically handled. This means that changing the scheduling will correspond to select new symbolic plans (or parts of plans) and to adopt new programs to control the equipment of the FMS.

A few alterations in an assembly component may imply to redesign other components or even redefine the way the assembly will be carried out. At present the Computer Aided Design (CAD) makes no account of that type of restrictions, the individual components project being carried out independently. The use of Solid Modeling with constraint propagation between components as well as some learning techniques are some possibilities to support Intelligent Design.

REFERENCES

[1] Rocha, J. and Ramos, C - *Task and Execution Planning for Flexible Manufacturing Systems*, Proc. of the Second International Conference on Integrated Logistics and Concurrent Engineering, Montpellier,France, 1994

[2] Rocha, J. and Ramos, C - *Generating and Converting Plans for Intelligent Manufacturing Systems*, Proc. of IEEE International Symposium on Industrial Electronics, Santiago, Chile, 1994

[3] Bourjault, A. - *Contribuition a une approche méthodologique de l'assemblage automatisé: Elaboration automatique des séquences opératoires*, Thèse d'État, Univ. de Franche-Comté, Besançon, France, 1984

[4] De Fazio T. and Whitney, D. - *Simplified generation of all mechanical assembly sequences*, IEEE Journal of Robotics and Automation, vol. 3, n. 6, pp. 640-658, December 1987

[5] Homem de Mello, L. - *Task sequence planning for robotic assembly*, PhD Dissertation, Carnegie Mellon University, 1989

[6] Wilson, R. - *On geometric assembly planning*, PhD Dissertation, Stanford University, Rep. n. STAN-CS-92-1416, 1992

[7] Lee, S. - *Backward Assembly Planning with assembly cost analysis*, Proc. of IEEE International Conference on Robotics and Automation, pp. 2382-2391, 1992

[8] Huang, Y. and Lee, C. - *A framework of knowledge-based assembly planning*, Proc. of IEEE International Conference on Robotics and Automation, pp. 599-604, 1991

[9] Lee, C. - *Proc. of the first workshop on assembly planning: theory and implementation* Workshop of IEEE International Conference on Robotics and Automation 1992

[10] Lozano-Pérez, T.; Jones, J.; Mazer, E.; O'Donnel, P. - *Task level planning of pick-and-place robot motions*, IEEE Computer, vol. 22, n. 3, pp. 21-29, March 1989

[11] Hutchinson, S. and Kak, A. - *SPAR: A planner that satisfies operational and geometric goals in uncertain environments*, AI Magazine, Spring 1990, pp. 30-61, 1990

[12] Ramos, C. and Oliveira, E. - *Closing the loop of Task Planning, Action and Sensing*, Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 909-916, Raleigh, NC, USA, 1992

[13] Ramos, C. and Oliveira, E. - *Sensor-based Reactive Planning and Execution for Robotics Assembly Tasks*, Proc. of IEEE International Conference on Systems Engineering, pp. 135-138, Kobe, Japan, 1992

[14] Ramos, C. and Oliveira, E. - *CIARC: A Multi-Agent Community for Intelligent Assembly Robotics Systems with Real-Time Capabilities*, Proc. of IEEE International Conference on Systems, Man and Cybernetics, Le Touquet, France, 1993

[15] Strobel, R. and Johnson, A. - *Pocket pagers in lots of one*, IEEE Spectrum - Manufacturing à la Carte, pp. 29-32, September 1993

[16] Kusiak, A. - *Intelligent Manufacturing Systems*, Prentice-Hall International Series in Industrial and Systems Engineering, 1990

DOCID: <XP__10141834A__I_>